January 1996

UILU-ENG-96-2201
CRHC-96-01

Center for Reliable and High Performance Computing

Do Not Take

State Justification Using Genetic Algorithms in Sequential Circuit Test Generation

Elizabeth M. Rudnick and Janak H. Patel

Last A half Copy

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-95-2201CRHC-96-01	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (*If applicable*) N/A
	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation ARPA
6c. ADDRESS (*City, State, and ZIP Code*) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (*City, State, and ZIP Code*) Research Triangle Park, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a	8b. OFFICE SYMBOL (*If applicable*)
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)

State Justification using Genetic Algorithms in Sequential Circuit Test Generation

12. PERSONAL AUTHOR(S)
Elizabeth M. Rudnick and Janak H. Patel

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	January 1996	20

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	genetic algorithms, sequential circuits, state justificatior test generation, VLSI circuit testing

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Complex VLSI circuits impose constraints on a test generator which are very difficult to handle using deterministic algorithms. Thus, a major goal in developing a new test generator is to have the capability of handling constraints, but without sacrificing the performance and effectiveness of deterministic approaches. In this paper, we describe a hybrid sequential circuit test generator which combines deterministic algorithms for fault excitation and propagation with genetic algorithms for state justification. The hybrid test generator restricts state justification for complex circuits to the genetic approach, which is better able to handle constraints. High fault coverages were obtained for the ISCAS89 sequential benchmark circuits and several synthesized circuits, and in many cases the results are better than those for purely deterministic approaches. Results were further augmented by preceding the hybrid test generation by a fast run of simulation-based test generation controlled by a genetic algorithm.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

State Justification using Genetic Algorithms in Sequential Circuit Test Generation *

Elizabeth M. Rudnick and Janak H. Patel

Center for Reliable and High-Performance Computing
University of Illinois
1308 West Main Street
Urbana, IL 61801

Abstract

Complex VLSI circuits impose constraints on a test generator which are very difficult to handle using deterministic algorithms. Thus, a major goal in developing a new test generator is to have the capability of handling constraints, but without sacrificing the performance and effectiveness of deterministic approaches. In this paper, we describe a hybrid sequential circuit test generator which combines deterministic algorithms for fault excitation and propagation with genetic algorithms for state justification. The hybrid test generator restricts state justification for complex circuits to the genetic approach, which is better able to handle constraints. High fault coverages were obtained for the ISCAS89 sequential benchmark circuits and several synthesized circuits, and in many cases the results are better than those for purely deterministic approaches. Results were further augmented by preceding the hybrid test generation by a fast run of simulation-based test generation controlled by a genetic algorithm.

*This research was supported in part by ARPA under Contract DABT63-95-C-0069, in part by the Semiconductor Research Corporation under Contract SRC 94-DP-109, and in part by Hewlett-Packard under an equipment grant.

I INTRODUCTION

Considerable progress has been made in deterministic sequential circuit test generation [1-7]. In a typical deterministic algorithm, each target fault is excited and the fault effects are propagated to a primary output (PO); the required state is then justified using reverse time processing. State justification involves backtracing through components and time. This approach is not easily adaptable to complex design features and tester constraints, however. For example, the initial state may be prespecified, a mix of scan and nonscan sequences may be desired in a partial scan environment, the tester may require that the values on certain pins be held constant for some number of clock cycles, power constraints may impose a limit on the number of simultaneous pin transitions, and so on. Previous work has shown that a simulation-based approach is capable of handling such constraints, for example, generation of mixed scan/nonscan sequences [8]. In a simulation-based approach, processing occurs in the forward direction only, and no backtracing is required. Therefore, complex component types are handled more easily. Candidate tests are generated, and a logic or fault simulator is used to select the best test to apply in a given time frame. Several faults are typically targeted simultaneously. Seshu and Freeman [9] first proposed simulation-based test generation, and several simulation-based test generators have since been developed using random [10], weighted random [11-13], and mutation-based [14, 15] pattern generators. Simulation-based test generators which use genetic algorithms (GAs) to generate candidate tests have also been developed [16-19]; very high fault coverages and fast execution times have been reported for several circuits.

A comparison of results for deterministic and GA-based test generators shows that each approach has its own merits. For some circuits, deterministic test generators provide higher fault coverages, while for other circuits, GA-based test generators provide higher fault coverages. The simulation-based approach is particularly well suited for data-dominant circuits, while deterministic test generators are more effective for control-dominant circuits. Untestable faults can be identified by using deterministic algorithms, but significant speedups can be obtained with the genetic approach. Hence, combining the two approaches could be beneficial. A straightforward solution would be to start with a fast run of the GA-based test

generator and then to use a deterministic test generator to improve the fault coverage and to identify untestable faults. Saab's CRIS-hybrid test generator [20] switches from simulation-based to deterministic test generation when a fixed number of test vectors are generated without improving the fault coverage; simulation-based test generation resumes after a test sequence is obtained from the deterministic procedure. We will explore a different approach which uses deterministic algorithms for fault excitation and propagation, and a GA for state justification. Individual faults in a circuit are targeted, as is normally done in deterministic test generators.

Deterministic algorithms for combinational circuit test generation have proven to be more effective than genetic algorithms [18]. Higher fault coverages are obtained, and the execution time is significantly smaller. A hybrid test generator would then naturally include the deterministic algorithm for fault excitation and propagation within a single time frame. Since we have access to the HITEC [6] source code, we also chose to use the deterministic algorithms for fault propagation in successive time frames, although GAs might be very useful for this purpose, and this approach is the subject of current research. State justification using deterministic algorithms is a much more difficult problem, however, especially if design and tester constraints are considered. In our hybrid test generator, we use a simulation-based approach for state justification in which candidate sequences evolve over several generations, as controlled by a GA. When a sequence which justifies the desired state is found, execution of the GA terminates. Our goal in this work is to show that this approach is a viable approach for future complex circuits. We do this by comparing its effectiveness with a deterministic approach. Results will show that they are indeed comparable.

We begin with a brief description of GAs and simulation-based test generation using a GA in Section II. An overview of our hybrid approach to test generation is given next in Section III, followed by a discussion of the application of GAs to state justification in Section IV. Results for the hybrid test generator are then presented in Section V for the ISCAS89 sequential benchmark circuits [21] and several synthesized circuits. In evaluating the hybrid approach, we also conducted experiments in which a fast run of a GA-based test generator is followed by either hybrid test generation or deterministic test generation. Both HITEC and

the GA-HITEC hybrid test generators were used in order to compare the various approaches. Results of these experiments are given in Section VI, and conclusions follow in Section VII.

II PRELIMINARIES

The simple GA, as described by Goldberg [22], contains a population of *strings,* or individuals. Each string is an encoding of a solution to the problem at hand. Each individual has an associated *fitness,* which gives an indication of the quality of the corresponding solution and thus depends on the application. The population is initialized with random strings, and the evolutionary processes of *selection, crossover,* and *mutation* are used to generate an entirely new population from the existing population. This process is repeated for a set number of generations or until no more improvements are obtained. To generate a new population from the existing one, two individuals are selected, with selection biased toward more highly fit individuals. The two individuals are crossed to create two entirely new individuals, and each character in a new string is mutated with some small mutation probability p. The two new individuals are then placed into the new population, and this process continues until the new generation is entirely filled. At this point, the previous generation can be discarded. In our work, we use tournament selection without replacement and uniform crossover. In *tournament selection without replacement,* two individuals are randomly chosen and removed from the population, and the best of the two is selected; the two individuals are not replaced into the original population until all other individuals have also been removed. Since two individuals are removed from the population for every individual selected, and the population size remains constant from one generation to the next, the original population is restored after the new population is half-filled. Therefore, the best individual will be selected twice, and the worst individual will not be selected at all. The number of copies selected of any other individual cannot be predicted except that it is either zero, one, or two. In *uniform crossover,* characters from the two parents are swapped with probability 1/2 at each string position in generating the two offspring. A crossover probability of 1 is used; i.e., the two parents are always crossed in generating the two offspring. Also, a mutation probability of 1/64 is used. Because selection is biased toward more highly fit individuals, the average

fitness is expected to increase from one generation to the next. However the best individual may appear in any generation.

In our previous work [18, 19], we described simulation-based test generators which used GAs to control the selection of candidate tests. Each individual in the GA population represents a test vector or a sequence of test vectors, and the PROOFS sequential circuit fault simulator [23] is used to evaluate the fitness of each candidate test. The fitness function used varies depending on the phase of test generation, but among the factors included, the highest weighting is given to the number of faults detected by a candidate test. This weighting encourages the evolution of tests which detect a large number of faults. The test generator begins by generating individual test vectors. Then test sequences are generated until no more improvements in fault coverage are made, at which point test generation terminates. During test sequence generation, various test sequence lengths are used, typically one, two, and four times the sequential depth of the circuit. Note that the *structural* sequential depth is used, rather than the *logical* sequential depth. The structural sequential depth of a circuit is the minimum number of flip-flops in a path between the primary inputs (PIs) and the furthest gate. For example, the structural sequential depth of the most significant bit of an n-bit binary counter is just n, but the logical sequential depth is 2^n. Experiments were conducted using various GA parameters [19], but high fault coverages were obtained using eight generations and a population size of 32.

If the GA-based test generator is to be used in a fast run prior to deterministic test generation, speedups can be obtained by eliminating the phase which generates individual test vectors, reducing the population size to 16, reducing the test sequence lengths by a factor of four, and using 100-fault samples in the fitness evaluations.

III OVERVIEW

Test generation using our hybrid approach is illustrated in Figure 1. An individual fault in the circuit is targeted. The fault is excited, and required values are backtraced to the PIs and flip-flops. Next, the fault effects are propagated to a PO, either in the current time frame or in successive time frames. Again, required values are backtraced to the PIs and

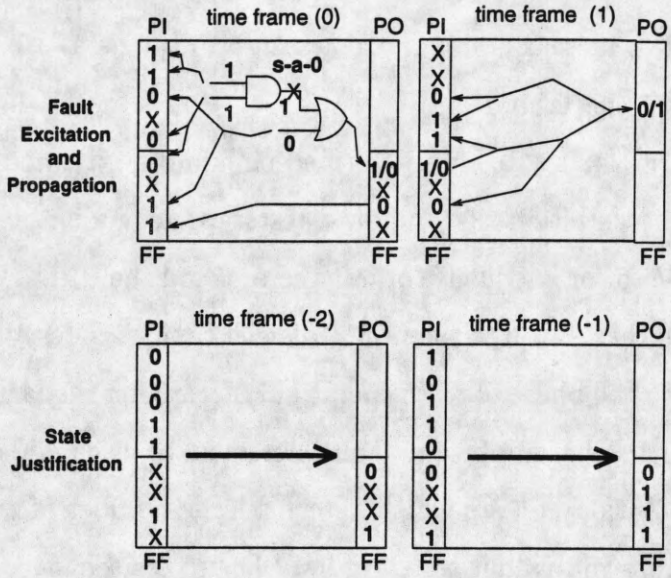

Figure 1: Test generation using GA for state justification.

to flip-flops in time frame zero, in which the fault was excited. If any conflicts are found during fault excitation and propagation, the test generator backtracks to a decision point and makes an alternative choice. Finally the required state in time frame zero is justified by using GAs. Several candidate sequences are simulated, starting from the last state reached after any previous tests have been applied. If a sequence is found which justifies the state, then the sequence is added to the test set, along with the vectors required for fault excitation and propagation. If a sequence cannot be found to justify the desired state, then backtracks are made in the fault propagation phase, and attempts are made to justify any new state.

One drawback to this approach is that untestable faults cannot be identified during state justification. Even if a sequence exists which justifies a given state, the GA is not guaranteed to find it. Therefore, deterministic algorithms for state justification are still required in a complete test generator. Hence, our overall approach to test generation includes both genetic and deterministic approaches for state justification, as indicated in Table I. The test generator makes several passes through the fault list, with different conditions and time limits imposed in each pass. Faults are removed from the fault list once they are detected. After each pass, the user is prompted as to whether to continue with another pass, and execution terminates when the user responds negatively.

Table I: GA-HITEC HYBRID TEST GENERATION

Pass	Test Generation Approach			Time Limit per Fault
I	Fault Excitation and Propagation with HITEC	⇒	State Justification with GA Population: 64 Generations: 4 Seq. length: 1/2x	0.5 s
II	Fault Excitation and Propagation with HITEC	⇒	State Justification with GA Population: 128 Generations: 8 Seq. length: x	5 s
III (optional)	Fault Excitation, Fault Propagation, and State Justification with HITEC			50 s and greater

In the first pass through the fault list, state justification is performed using a GA. A time limit of one-half second per fault is imposed, but the time is checked at backtrack decision points only; i.e., the GA evolves over four full generations before the time is checked. Therefore, the actual time spent per fault could be greater than one-half second. A small population size of 64 is used, and the number of generations is limited to four to reduce the execution time. A sequence length of $\frac{1}{2}x$ is used, where x is supplied by the user. Many of the testable faults are detected in this pass, but untestable faults are identified only if conflicts are found without doing state justification. In the second pass through the fault list, GAs are again used for state justification, but the search space is expanded. In particular, the population size is increased to 128, the number of generations is increased to eight, and the sequence length is doubled. Also, the time limit is increased to 5 seconds per fault to enable more backtracking in the fault propagation phase. Phase III is an optional phase in which deterministic algorithms are used for state justification for any additional passes through the fault list. Depending on the circuit constraints, the option of deterministic state justification may not be possible. Required values at the flip-flops are backtraced to the PIs in previous time frames through reverse time processing. An untestable fault is identified when all possible choices at decision points prove unsuccessful in generating a test to detect the fault. The time limit per fault is increased to 50 seconds in the third pass and multiplied by ten in successive passes to expand the search space. The time is checked before each

new time frame is processed, as well as at backtrack decision points. In this manner, tests are generated for many of the testable faults by using the GA for state justification. The deterministic algorithms for state justification are used to identify untestable faults and to generate tests for hard-to-detect faults only.

IV APPLICATION OF GA'S TO STATE JUSTIFICATION

In applying GAs to state justification, we use each string in the population to represent a candidate test sequence. A binary coding is used, and successive vectors in the sequence are placed in adjacent positions along the string. Sequences are evolved over several generations, with the fitness of each individual being a measure of how closely the final state reached matches the desired state. If any sequence is found which produces the desired state, the search is terminated, and the sequence is added to the test set, along with the fault excitation and propagation vectors. Otherwise, the GA runs to completion for a limited number of generations. The test sequence length used is typically a multiple of the structural sequential depth of the circuit.

A Fitness Function

Since the fitness of an individual sequence indicates how closely the state it produces matches the desired state, simulation is required. The presence of a particular fault may affect the state; thus, both good and faulty circuit simulations are required for an accurate result. If tests have already been added to the test set, then the current good circuit flip-flop values may already be known. However, the state is not known for the faulty circuit unless this information is retrieved from the fault simulator, as is done by Kelsey, Saluja, and Lee in FASTEST [24]. Instead of retrieving the faulty circuit state, we initialize the faulty circuit flip-flops to unknown values. Using a known faulty circuit state might improve the chances of finding a state justification sequence and might also result in more compact test sets.

Before the search is begun for a sequence to justify a required state, the desired good circuit state is compared to the current good circuit state, and the desired faulty circuit state is compared to the all-unknown state. (Note that separate values are maintained for the good and faulty circuits during the fault excitation and propagation phases.) If the states

match, no justification is required. If the current state does not match the desired state, then several candidate sequences are simulated for both the good and faulty circuits. Fault injection is performed by modifying the circuit description, as is done in PROOFS [23]; eg., an OR-gate is inserted to simulate a stuck-at-one fault, and the second input of the OR-gate is set to zero for the good circuit and one for the faulty circuit. The bitwise parallelism of the computer word is used, which allows 32 sequences to be simulated in parallel. Two bits are required to represent the three possible logic values: one, zero, and X (unknown). Thus, two computer words are used at each node to simulate the good circuit, and two computer words are used at each node to simulate the faulty circuit. PI values are mapped from the sequences in the GA to the respective bit positions at the PI nodes. Simulation is done in an event-driven manner, with good and faulty circuit simulations done together.

The test sequence length is set to a fixed value, but the state is checked after each test vector is simulated to determine whether it matches the desired state. If it does, the search is terminated, and the vectors which enable the circuit to reach the desired state are added to the test set, along with the fault excitation and propagation vectors. Therefore, the length of the actual test sequence used may be less than the given value. However, for the purposes of the GA, the fitness function measures how closely the final state produced by a test sequence matches the desired state:

$$\begin{aligned} fitness \;=\; & \frac{9}{10}(number\ of\ matching\ flip\ flops\ in\ good\ circuit) \\ & + \frac{1}{10}(number\ of\ matching\ flip\ flops\ in\ faulty\ circuit). \end{aligned}$$

A flip-flop is considered to match if it requires no particular value or if the desired and actual values are equal. If the states match in both the good and faulty circuits, then the fitness will equal the number of flip-flops in the circuit. The two terms in the fitness function correspond to the two goals of the GA: finding a state justification sequence for the good circuit and finding a state justification sequence for the faulty circuit. Unequal weights are used in order that the GA can be targeted to one goal at a time. When a GA has two or more goals, the optimum fitness function does not necessarily weight the goals equally. If equal weights are used, the GA jumps back and forth among the goals, and none of the

problems gets solved quickly. A heavy weighting of one goal ensures that the strings evolve steadily in one direction. Experiments on several circuits confirmed that the weights chosen work better than equal weights of 1/2.

Squaring of the fitness function has been used previously to amplify the differences between individuals [22]. Such a measure should be considered if a proportionate selection scheme is used. However, this operation would have no effect with tournament selection, which is used in our GA, since the best of two randomly chosen individuals is always selected, no matter how large the difference in fitness values between the two.

B GA Parameters

Since 32 sequences can be evaluated in parallel, the population size should be a multiple of 32. Initially, we use a small population size of 64 to limit the execution time. We increase it to 128 in the second pass through the fault list, expanding the search space. The number of generations is initially limited to four, again to reduce the execution time. We increase the number of generations to eight in the second pass, when expanding the search space. Tournament selection and uniform crossover are used, since these schemes worked well in simulation-based test generation [19]. Crossover and mutation probabilities of one and 1/64, respectively, are used. Nonoverlapping generations are used, since exploration of the search space is paramount.

V GA-HITEC RESULTS

A hybrid test generator, GA-HITEC, was implemented using the existing HITEC [6] source code and 2700 additional lines of C++ code. Tests were generated for several of the ISCAS89 sequential benchmark circuits [21] on an HP 9000 J200 with 256 MB memory. Test generation results are shown in Tables II and III. Results for HITEC are shown for comparison. The three lines of results for each circuit correspond to three passes through the fault list with time limits and parameter settings as shown in Table I for GA-HITEC. One exception is that a population size of 32 was used for passes one and two for circuit s35932 to speed up the execution. Test sequence lengths of four and eight times the structural sequential depth were used in passes one and two, respectively, for all circuits except s5378 and s35932.

Test sequence lengths were one-quarter and one-half the sequential depth for these circuits. Higher fault coverages might be obtained with longer test sequences, but the execution time would increase. HITEC also makes several passes through the fault list. The time and backtrack limits are initially set to one-half second and 10,000 backtracks, respectively, and they are multiplied by ten in each successive pass. Further improvements in fault coverage and untestable fault identification are possible for both GA-HITEC and HITEC if a fourth pass is made through the fault list using a time limit of 500 seconds per fault; however, execution times would increase. The number of faults detected (**Det**), the number of test vectors generated (**Vec**), the total execution time, and the number of untestable faults identified (**Unt**) at the end of each pass are shown for both GA-HITEC and HITEC. The best results are highlighted in the tables, including the highest fault coverages after each of the three passes and also the smallest test set sizes and execution times after the third pass for cases in which the fault coverage is the maximum.

For many circuits, more faults are detected by GA-HITEC than HITEC at the end of each of the first two passes. In most cases, the GA-HITEC fault coverage at the end of the third pass is greater than or equal to that of HITEC. These results show that the GA is effective in searching for state justification sequences, especially when it is combined with the deterministic approach. In the first two passes, GA-HITEC is able to make use of the current good circuit state, i.e., the state reached after all previous sequences in the test set have been applied. In contrast, HITEC always backtraces to a time frame in which all flip-flops are set to unknown (don't care) values. However, GA-HITEC is not a superset of HITEC. Although GA-HITEC uses the same algorithms as HITEC after the second pass, the HITEC fault coverage is sometimes higher after the third pass. For example, HITEC detects 34,901 faults in circuit s35932 after the third pass, while GA-HITEC detects only 34,879 faults. This discrepancy occurs because the algorithms used in HITEC are partially nondeterministic. Many of the faults are incidentally detected by the test sequences generated; these faults are identified by the fault simulator, and they are never targeted by the test generator. The fault coverage thus depends on the fault list ordering and the time limit imposed on each fault.

Table II: GA-HITEC TEST GENERATION RESULTS: SMALLER CIRCUITS

Circuit	Seq Depth	Total Faults	GA-HITEC				HITEC			
			Det	Vec	Time	Unt	Det	Vec	Time	Unt
s298	8	308	*262*	264	24.0s	0	253	110	24.0s	21
			264	330	3.29m	0	*265*	*322*	*2.01m*	26
			265	354	17.7m	26	265	*322*	*16.2m*	26
s344	6	342	*328*	153	9.85s	0	321	93	9.44s	9
			328	153	*1.11m*	0	321	93	1.17m	9
			328	153	*4.01m*	11	324	115	8.07m	11
s349	6	350	*335*	169	*7.93s*	2	323	86	11.7s	11
			335	169	*58.2s*	2	324	94	1.51m	11
			335	169	*3.02m*	13	332	128	7.73m	13
s382	11	399	53	35	3.56m	0	*61*	89	*3.34m*	0
			309	*399*	13.7m	0	290	974	13.2m	3
			328	*914*	*1.22h*	9	301	1463	1.52h	9
s386	5	384	284	198	34.6s	0	*314*	286	*7.25s*	70
			296	302	3.57m	0	*314*	*286*	*7.25s*	70
			314	384	3.65m	70	314	*286*	*7.25s*	70
s400	11	426	*75*	*80*	3.45m	6	65	94	3.29m	6
			338	*370*	*10.7m*	6	331	1532	11.6m	9
			343	*741*	*1.16h*	17	341	1845	1.21h	17
s444	11	474	57	66	4.30m	14	*68*	103	*3.87m*	15
			401	*744*	*9.97m*	14	319	971	17.3m	17
			403	*928*	*52.4m*	24	373	1761	1.49h	25
s526	11	555	*71*	121	6.45m	1	51	34	5.94m	7
			376	851	*25.6m*	2	51	34	51.7m	16
			388	1096	*2.61h*	20	316	436	5.79h	23
s641	6	467	404	269	16.8s	41	404	*209*	*4.84s*	63
			404	269	1.63m	41	404	*209*	*4.84s*	63
			404	269	1.66m	63	404	*209*	*4.84s*	63
s713	6	581	476	242	18.8s	82	476	*173*	*6.71s*	105
			476	242	2.54m	82	476	*173*	*6.71s*	105
			476	242	2.59m	105	476	*173*	*6.71s*	105
s820	4	850	446	267	3.36m	0	*765*	773	*1.31m*	19
			479	408	25.5m	0	*812*	1103	*2.58m*	34
			814	*1084*	27.2m	36	813	1115	*3.50m*	37
s832	4	870	449	297	3.46m	14	*598*	351	*3.12m*	35
			483	397	25.8m	14	*816*	1125	*4.77m*	51
			818	*1054*	28.3m	52	817	1137	*5.75m*	53

Det: # faults detected Vec: # test vectors generated Unt: # untestable faults identified

Table III: GA-HITEC TEST GENERATION RESULTS: LARGER CIRCUITS

Circuit	Seq Depth	Total Faults	GA-HITEC				HITEC			
			Det	Vec	Time	Unt	Det	Vec	Time	Unt
s1196	4	1242	1238	381	7.86s	3	*1239*	435	*5.50s*	3
			1239	*383*	11.4s	3	1239	435	*5.50s*	3
s1238	4	1355	1283	*386*	10.1s	72	1283	475	*8.23s*	72
s1423	10	1515	*760*	387	12.0m	9	554	96	9.71m	10
			794	499	*1.48h*	10	554	96	1.51h	11
			893	541	*10.8h*	14	723	150	13.9h	14
s1488	5	1486	*1180*	346	*3.91m*	0	797	69	7.24m	7
			1225	576	24.8m	0	*1439*	1136	*12.5m*	29
			1444	1272	30.2m	41	1444	*1170*	*16.5m*	41
s1494	5	1506	*1166*	370	4.22m	12	1132	311	3.72m	20
			1259	608	24.6m	12	*1452*	1237	*6.94m*	42
			1453	1315	29.2m	52	1453	*1245*	*9.59m*	52
s3271	9	3270	*3225*	780	2.46m	0	3195	503	56.8s	1
			3228	810	14.8m	0	3222	577	5.43m	3
			3250	952	*31.3m*	5	3227	641	39.4m	5
s3330	8	2870	2086	1175	22.7m	16	*2088*	*492*	*7.82m*	16
			2098	1362	3.33h	16	2095	532	1.18h	63
			2119	1393	12.4h	124	2097	551	10.6h	124
s3384	9	3380	*3189*	1044	13.4m	0	2953	141	4.31m	0
			3205	1207	1.11h	0	2996	161	40.4m	1
			3205	1207	*3.56h*	1	2996	161	6.06h	1
s4863	5	4764	4504	381	9.41m	0	*4512*	*267*	*3.56m*	1
			4505	397	52.8m	0	*4571*	*374*	*22.6m*	1
			4609	562	3.21h	22	*4621*	*477*	*2.38h*	25
s5378	36	4603	2996	337	21.2m	74	*3231*	912	*13.4m*	122
			3193	492	3.55h	78	*3231*	912	*1.97h*	164
			3239	*623*	19.9h	225	3231	912	18.4h	217
s6669	9	3380	6600	264	12.7m	0	*6622*	*251*	*1.41m*	0
			6615	337	1.03h	0	*6643*	*286*	*5.80m*	0
			6657	429	1.57h	0	6655	319	33.3m	0
s35932	35	39094	33,341	285	2.91h	3856	*34,170*	287	*1.29h*	3728
			33,766	391	7.19h	3984	*34,901*	496	*1.78h*	3984
			34,879	568	10.5h	3984	*34,901*	*496*	*4.73h*	3984

Det: # faults detected Vec: # test vectors generated Unt: # untestable faults identified

While fewer untestable faults are generally identified in the first two passes with GA-HITEC, approximately the same number are identified after the third pass. In some cases, such as circuit s832, HITEC declares one or two testable faults to be untestable. This incorrect identification occurs because PODEM [25] is used for processing the combinational logic within a time frame, and PODEM is susceptible to over-specification of the state required for fault excitation [26]. HITEC uses a heuristic to compensate for over-specifications, attempting to excite the fault after each required flip-flop value is temporarily set to unknown. However, incorrect untestable fault identifications are still possible [26]. Comparison of execution times shows that GA-HITEC is faster for some circuits, while HITEC is faster for other circuits. GA-HITEC wastes time targeting untestable faults in the first two passes, a result especially apparent for circuit s386. If these untestable faults can be filtered out in advance by a preprocessing program, such as the one developed by Iyer and Abramovici [27], significant speedups can be obtained.

Results of running GA-HITEC on several circuits synthesized from high-level descriptions are shown in Table IV. The **Am2910** circuit is a 12-bit microprogram sequencer similar to the one described in [28]; **div** is a 16-bit divider which uses repeated subtraction to perform division; **mult** is a 16-bit two's complement multiplier which uses a shift-and-add algorithm; **pcont2** is an 8-bit parallel controller used in DSP applications; and **piir8** is an 8-point infinite impulse response filter for DSP applications. Test sequence lengths of 24 and 48 were used in the first two passes through the fault lists. For larger circuits, smaller test sequence lengths could be used and the GA parameters could be adjusted to speed up the execution, but lower fault coverages might then be obtained. Results for HITEC are shown for comparison. The three lines of results for each circuit correspond to the individual passes through the fault list, and again the best results are highlighted. GA-HITEC yielded higher fault coverages than HITEC after each of the three passes for all five circuits, and the overall GA-HITEC execution times were also smallest for all five circuits.

In summary, fault coverages and execution times for GA-HITEC and HITEC are comparable after the first two passes through the fault list for most circuits. In some cases, HITEC performs better, while in other cases, GA-HITEC performs better. Fault coverages

Table IV: GA-HITEC TEST GENERATION RESULTS: SYNTHESIZED CIRCUITS

Circuit	Seq Depth	Total Faults	GA-HITEC				HITEC			
			Det	Vec	Time	Unt	Det	Vec	Time	Unt
Am2910	4	2391	*2163*	747	*1.70m*	157	1991	348	4.13m	157
			2175	880	*6.90m*	159	2130	585	12.8m	170
			2187	1002	*34.3m*	173	2171	871	56.4m	173
div	19	2147	*1722*	229	*4.93m*	136	1664	224	6.62m	136
			1722	229	*29.3m*	136	1664	224	37.9m	136
			1723	251	*4.39h*	136	1667	228	5.35h	136
mult	9	1708	*1548*	236	*3.55m*	3	1319	69	5.26m	8
			1550	285	*22.4m*	3	1487	90	23.6m	14
			1606	306	*1.56h*	23	1582	111	1.90h	23
pcont2	3	11,300	*6748*	174	*48.1m*	2651	3514	7	2.25h	2585
			6752	206	*4.59h*	2770	3514	7	9.58h	2773
			6752	206	*29.3h*	2801	3514	7	79.5h	2799
piir8	13	19,936	*11,504*	53	10.4h	3470	9003	21	3.62h	3662
			11,504	53	44.0h	4814	9003	21	13.1h	4817
			11,504	53	*94.7h*	4814	9003	21	98.8h	4817

Am2910: 12-bit microprogram sequencer
div: 16-bit divider
mult: 16-bit two's complement multiplier
pcont2: 8-bit parallel controller for DSP applications
piir8: 8-point infinite impulse response filter for DSP applications

Det: # faults detected
Vec: # test vectors generated
Unt: # untestable faults identified

for GA-HITEC after two passes are often better than those for HITEC after three passes. Our goal in this work was to show that a GA is capable of doing state justification, and the results indirectly show that GAs are indeed effective.

VI RESULTS FOR COMBINED APPROACHES

While the hybrid test generation approach is effective for benchmark circuits, it may be even more useful for real circuits from industry. Complex VLSI circuits are described at mixed levels of abstraction, including gates, functional primitives, high-level primitives, and behavioral descriptions, and they contain complex design features, such as multiple, derived, and gated clocks; mixed positive and negative clocking; a mix of latches and flip-flops; asynchronous logic; embedded RAMs, ROMs, and megacells; and switch-based custom logic. These complex design features impose contraints on the test generator which are

difficult to satisfy with deterministic approaches. Furthermore, the tester may also impose constraints, eg., requiring values on pins to be held constant for a given number of clock cycles. Also, it may be desirable to use a mix of scan and nonscan sequences in a partial scan environment in order to reduce the test application time [8]; this approach imposes additional constraints. Backtracing is used during the fault excitation and propagation phases in the hybrid test generator, and the processing is restricted to the forward direction during state justification. The third pass of deterministic test generation is not available due to the constraints. However, not having access to Phase III is not a major handicap. Results show only minor drops in the fault coverages for the synthesized circuits without Phase III. A large number of the untestable faults are still identified, and significant speedups are obtained. The abbreviated hybrid test generation can be preceded with a fast run of a GA-based test generator, such as GATEST [19], to improve the fault coverage and execution time, and the combined approach is well-suited for complex VLSI circuits.

To evaluate the combined approach, experiments were conducted in which a fast run of GATEST was followed by either hybrid test generation or deterministic test generation. The GATEST parameters discussed in Section II were used to minimize the execution time. Test sequence lengths of 6, 12, and 24 were used for all circuits. Results are given in Table V for the synthesized circuits, with the best results highlighted. The same parameters described in Section V were used for hybrid and deterministic test generation, except that GA-HITEC was limited to two passes through the fault list.

Table V: RESULTS OF COMBINING GATEST AND GA-HITEC

	GATEST			**GATEST + GA-HITEC**				**GATEST + HITEC**			
Circuit	**Det**	**Vec**	**Time**	**Det**	**Vec**	**Time**	**Unt**	**Det**	**Vec**	**Time**	**Unt**
Am2910	2174	690	8.48m	2191	855	14.1m	159	*2192*	*743*	34.3m	173
div	1689	558	5.56m	*1726*	651	*44.8m*	136	1725	618	4.41h	136
mult	1621	144	3.43m	1621	144	*13.2m*	3	1621	144	1.13h	23
pcont2	6816	180	7.30m	6816	180	*4.40h*	2770	6816	180	26.9h	2805
piir8	15,017	432	23.8m	15,017	432	*2.36h*	4791	15,017	432	3.50h	4791

Det: # faults detected Vec: # test vectors generated Unt: # untestable faults identified

Significant improvements in fault coverage and execution time were obtained when the fast GATEST run was used. As a matter of fact, fault coverages obtained by GATEST alone are

almost as good as those reported previously when the GA parameters were tuned to optimize fault coverage rather than execution time [19]. Fault coverages for the GATEST/HITEC combination improved over those for HITEC alone, and fault coverages for the GATEST/GA-HITEC combination improved over those for GA-HITEC alone for all circuits after only two passes through the fault list. No additional faults were detected by HITEC in the third pass through the fault list, although more untestable faults were identified for some circuits. The number of faults detected for the GATEST/GA-HITEC combination was within one fault of the number detected by the GATEST/HITEC combination.

Execution times for the GATEST/GA-HITEC combination were significantly lower than those for the GATEST/HITEC combination, and a large fraction of the untestable faults were also identified. For mult, pcont2, and piir8, GATEST alone generates test vectors to cover testable faults, but GA-HITEC is still effective in identifying untestable faults. For the remaining synthesized circuits, GA-HITEC is able to generate additional test vectors to improve the fault coverage and also identify many untestable faults. Thus, the GATEST/GA-HITEC combination outperforms GATEST alone in terms of fault coverage and untestable fault identification. A comparison of fault coverages for the GATEST/GA-HITEC combination and the CRIS-hybrid [20] is difficult, since slightly different versions of some of the circuits appear to have been used. Nevertheless, for comparable fault coverages, the GATEST/GA-HITEC combination achieves significantly lower execution times for most circuits. Furthermore, it is better able to handle the constraints of complex VLSI designs, as discussed earlier.

VII CONCLUSIONS

Deterministic algorithms for fault excitation and propagation have been combined with a genetic algorithm for state justification in a new hybrid sequential circuit test generator, GA-HITEC. GA-HITEC makes several passes through the fault list, targeting individual faults, with time limits increasing in successive passes. GAs are used for state justification in the first two passes, while a deterministic algorithm is used in any additional passes. Results for the ISCAS89 benchmark circuits demonstrate the effectiveness of GAs for state

justification. Higher fault coverages are obtained for GA-HITEC as compared to HITEC for many circuits. Approximately the same number of untestable faults are identified for the two test generators, and GA-HITEC executes more quickly for many of the circuits.

While the hybrid test generation approach is effective for benchmark circuits, it may be even more useful for the complex VLSI circuits designed in industry. Real circuits may impose constraints on the test generator which are difficult to satisfy with deterministic approaches. The GA-HITEC hybrid approach can be used to restrict processing to the forward direction during state justification in order to handle the constraints of complex design features. Furthermore, GA-HITEC can be combined with a fast run of a simulation-based test generator to improve the fault coverage and execution time, while identifying a large fraction of the untestable faults.

Finally, this research can be extended to justification of module output values in architectural-level test generation. Backtracing required values through high-level modules is a difficult problem, but a genetic approach could be used in place of traditional approaches to simplify the test generator.

ACKNOWLEDGMENT

The authors would like to thank Prof. David Goldberg for providing several useful suggestions.

References

[1] R. Marlett, "An effective test generation system for sequential circuits," *Proc. Design Automation Conf.*, pp. 250-256, 1986.

[2] W. -T. Cheng, "The BACK algorithm for sequential test generation," *Proc. Int. Conf. Computer Design,* pp. 66-69, 1988.

[3] H. -K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test generation for sequential circuits," *IEEE Trans. Computer-Aided Design,* vol. 7, no. 10, pp. 1081-1093, October 1988.

[4] M. H. Schulz and E. Auth, "Essential: An efficient self-learning test pattern generation algorithm for sequential circuits," *Proc. Int. Test Conf.*, pp. 28-37, 1989.

[5] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation for highly sequential circuits," *Proc. Int. Conf. Computer-Aided Design,* pp. 362-365, 1989.

[6] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," *Proc. European Conf. Design Automation*, pp. 214-218, 1991.

[7] D. H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," *Proc. Int. Conf. Computer-Aided Design*, pp. 446-449, 1991.

[8] E. M. Rudnick and J. H. Patel, "A genetic approach to test application time reduction for full scan and partial scan circuits," *Proc. Eighth Int. Conf. VLSI Design*, pp. 288-293, 1995.

[9] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," *IRE Trans. Electronic Computing,* vol. 11, pp. 459-465, August 1962.

[10] M. A. Breuer, "A random and an algorithmic technique for fault detection test generation for sequential circuits," *IEEE Trans. Computers,* vol. 20, no. 11, pp. 1364-1370, November 1971.

[11] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The weighted random test-pattern generator," *IEEE Trans. Computers,* vol. 24, no. 7, pp. 695-700, July 1975.

[12] R. Lisanke, F. Brglez, A. J. Degeus, and D. Gregory, "Testability-driven random test-pattern generation," *IEEE Trans. Computer-Aided Design,* vol. 6, no. 6, pp. 1082-1087, November 1987.

[13] H.-J. Wunderlich, "Multiple distributions for biased random test patterns," *IEEE Trans. Computer-Aided Design,* vol. 9, no. 6, pp. 584-593, June 1990.

[14] T. J. Snethen, "Simulator-oriented fault test generator," *Proc. Design Automation Conf.*, pp. 88-93, 1977.

[15] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A directed search method for test generation using a concurrent simulator," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 2, pp. 131-138, February 1989.

[16] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," *Proc. Int. Conf. Computer-Aided Design*, pp. 216-219, 1992.

[17] M. Srinivas and L. M. Patnaik, "A simulation-based test generation scheme using genetic algorithms," *Proc. Int. Conf. VLSI Design*, pp. 132-135, 1993.

[18] E. M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of simple genetic algorithms to sequential circuit test generation," *Proc. European Design and Test Conf.*, pp. 40-45, 1994.

[19] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework," *Proc. Design Automation Conf.,* pp. 698-704, 1994.

[20] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Iterative [simulation-based genetics + deterministic techniques] = complete ATPG," *Proc. Int. Conf. Computer-Aided Design*, pp. 40-43, 1994.

[21] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Reading, MA: Addison-Wesley, 1989.

[23] T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 2, pp. 198-207, February 1992.

[24] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An efficient algorithm for sequential circuit test generation," *IEEE Trans. Computers*, vol. 42, no. 11, pp. 1361-1371, November 1993.

[25] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Computers*, C-30, pp. 215-222, March 1981.

[26] K.-T. Cheng and H.-K. T. Ma, "On the over-specification problem in sequential ATPG algorithms," *Proc. Design Automation Conf.*, pp. 16-21, June 1992.

[27] M. A. Iyer and M. Abramovici, "Sequentially untestable faults identified without search," *Proc. Int. Test Conf.*, pp. 259-266, 1994.

[28] Advanced Micro Devices, "The AM2910, a complete 12-bit microprogram sequence controller," in *AMD Data Book*, AMD Inc., Sunnyvale, CA, 1978.